Uppsala University
Department of Information Technology
Division of Systems and Control
DW/NW 2019-06
Last rev. September 30, 2020 by DW

Advanced Probabilistic Machine Learning

Instruction to the laboratory work

Unsupervised Learning with Variational Autoencoders

Language: Python

Preparation:
Solve all preparatory exercises in Section 3

Reading instructions:

- Laboratory instructions: Section 1-3.
- Kingma, D. P., & Welling, M. (2019). *An introduction to variational autoencoders*. arXiv: 1906.02691 `[cs.LG]`.

Name		Assistant's comments
Program	Year of reg.	
Date		
Passed prep. ex.	Sign	
Passed lab.	Sign	

Contents

1 Introduction

Unsupervised learning tries to find unknown patterns in unlabeled data. In the lecture we have seen principal component analysis (PCA), probabilistic principal component analysis (PPCA), and variational autoencoders as three examples of unsupervised learning. In this lab we will try to find patterns in images of handwritten digits and train a model that can generate similarly looking images.

This laboratory work is based on lectures 9 and 10 about unsupervised learning and variational autoencoders. Therefore it is advisable to have the material from those lectures fresh in mind before starting this laboratory work.

The goal of this laboratory work is to:

- Learn how to perform a PCA.
- Learn how to train a probabilistic PCA model.
- Learn how to train a variational autoencoder.
- Get a glimpse of a state-of-the-art machine learning framework.

Throughout the lab we will us a software library called *PyTorch*. This library is introduced in Section 2. Section 3 contains the preparatory exercises, and Section 4 contains the instructions for the lab session.

Important: Read Section 2 and answer the preparatory exercises in Section 3 *before* the lab session.

2 PyTorch

PyTorch is an open source software library for machine learning that is based on the machine learning library Torch which is no longer actively developed. PyTorch is developed primarily by Facebook's artificial intelligence research group and used by companies as well as academic research groups. It can be used for general computations with multidimensional arrays on CPUs and GPUs, but it is tailored especially to deep learning and neural networks.

PyTorch is natively written in Python and C++, and well documented APIs exist for these languages. It is not the only deep learning framework, some other state-of-the-art alternatives are TensorFlow and MXNet.

2.1 Installation

PyTorch is already installed on the Linux systems in the computer rooms where the laboratory session is scheduled. You can either use these computers during the lab or bring your own computer.

If you choose to use your own computer, you need to have PyTorch properly installed *before* the lab. The lab assistants will not be able to assist you with the installation process during the lab. Please consult the PyTorch documentation for more information about the installation procedure.

Another option is to use Google Colab to work with PyTorch online. This cloud platform also optionally provides access to GPUs which might speed up some computations.

2.2 Comparison with TensorFlow

As PyTorch, TensorFlow contains support for GPUs and builds a computational graph that is used for computing gradients via backpropagation. However, Tensorflow builds a static computational graph whereas PyTorch works with a dynamic computational graph. In TensorFlow the graph is created once and then executed multiple times, but in PyTorch every forward pass defines a new computational graph. Thus in TensorFlow the computational graph can be optimized before running any actual computations. Even if this optimization might be computationally expensive, it can pay off if the graph is run multiple times. On the other hand, an advantage of using dynamic graphs is the increased flexibility with respect to the control flow that they allow. Since the computational graph is rebuilt in each forward pass, one can use regular `if` statements and `for` loops (even without fixed numbers of iterations) and easily change the computations in different iterations of the training loop.

2.3 Introduction

A Jupyter notebook `introduction.ipynb` with an introduction to PyTorch can be downloaded from studium. Reading and running the notebook is highly recommended, since it introduces important concepts and commands that are required in the lab session.

The official PyTorch tutorials and the PyTorch documentation might be helpful additional resources, in particular if you are looking for a more general introduction to PyTorch. However, the introduction to PyTorch in the provided Jupyter notebook covers everything you should know for the lab session.

3 Preparatory exercises

3.1 Principal component analysis

In the lectures the principal component analysis (PCA) was introduced as a method for dimensionality reduction and feature extraction, i.e., to condense data by mapping it to a lower dimensional space of the most important features.

Let

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\intercal \\ \vdots \\ \mathbf{x}_N^\intercal \end{bmatrix} \in \mathbb{R}^{N \times D}$$

be a matrix of N data samples $\mathbf{x}_n \in \mathbb{R}^D$, which are centered around zero, i.e., $\frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n = \mathbf{0}$. We consider a PCA with $M < D$ components.

To project the data points $\mathbf{x}_n$ to the M-dimensional space that is defined by the M principal components of $\mathbf{X}$, the so-called principal subspace of $\mathbf{X}$, we can use the singular value decomposition of $\mathbf{X}$. Let $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$ be the singular value decomposition of the data matrix $\mathbf{X}$ with the singular values sorted in descending order.[1] Then the projection $\mathbf{z}_n$ of data point $\mathbf{x}_n$ to the principal subspace of $\mathbf{X}$ is given by

$$\mathbf{z}_n^\intercal = \mathbf{x}_n^\intercal \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_M \end{bmatrix}, \tag{1}$$

where $\mathbf{v}_i$ is the ith column of matrix $\mathbf{V}$. The vector $\mathbf{z}_n$ can be seen as an "encoding" of the data point $\mathbf{x}_n$ in a lower dimensional space that is constructed by the directions for which the data shows the largest variations.

Ideally we would like to be able to recover (or "decode") the original data from the lower dimensional encodings as good as possible. With the help of the singular value decomposition of $\mathbf{X}$, we can compute reconstructions $\tilde{\mathbf{x}}_n$ from the encodings $\mathbf{z}_n$ by

$$\tilde{\mathbf{x}}_n^\intercal = \mathbf{z}_n^\intercal \begin{bmatrix} \mathbf{v}_1^\intercal \\ \vdots \\ \mathbf{v}_M^\intercal \end{bmatrix}. \tag{2}$$

Since PCA can be seen as a mapping of the data from the original coordinate system to the principal subspace, we can map also other data points to the principal subspace that we have learnt from data set $\mathbf{X}$.

Question 3.1: *How can you map an arbitrary vector* $\mathbf{x} \in \mathbb{R}^D$ *to the principal subspace of* $\mathbf{X}$ *with the help of* $\mathbf{U}$, $\mathbf{\Sigma}$, *and* $\mathbf{V}$*? Just provide the mathematical formula.*

[1]The singular value decomposition of a matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ is defined as a factorization of the form $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$ where $\mathbf{U} \in \mathbb{R}^{N \times N}$ and $\mathbf{V} \in \mathbb{R}^{D \times D}$ are orthogonal matrices and $\mathbf{\Sigma} \in \mathbb{R}^{N \times D}$ is a rectangular diagonal matrix with non-negative numbers on the diagonal. The diagonal entries of $\mathbf{\Sigma}$ are the so-called singular values of $\mathbf{X}$. A common convention is to sort the singular values in descending order, in which case the diagonal matrix $\mathbf{\Sigma}$ is uniquely determined by $\mathbf{X}$.

Answer:

Often the data $\mathbf{X}$ is not centered around zero, i.e., $\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \neq \mathbf{0}$.

Question 3.2: *How do you have to change the calculations in eqns.* (1) *and* (2) *and Question 3.1 in such a case?*

Answer:

3.2 Probabilistic principal component analysis

In constrast to (regular) PCA, the so-called probabilistic PCA (PPCA) (Tipping & Bishop, 1999) allows a probabilistic interpretation of the principal components. The probabilistic formulation of PCA also allows us to extend the method and alter its underlying assumptions quite easily, as we will learn during the course of this laboratory.

As before, let $\mathbf{x} \in \mathbb{R}^D$ represent a data sample that we want to decode from a lower dimensional representation $\mathbf{z} \in \mathbb{R}^M$ with $M < D$. The PPCA model assumes that $\mathbf{z}$ is standard normally distributed and $\mathbf{x}$ can be decoded by a noisy linear transformation of $\mathbf{z}$. Mathematically, the model is given by

$$\begin{aligned} p(\mathbf{x} \,|\, \mathbf{z}) &= \mathcal{N}\left(\mathbf{x}; \mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_D\right), \\ p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}_M), \end{aligned}$$

with parameters $\mathbf{W} \in \mathbb{R}^{D\times M}$, $\boldsymbol{\mu} \in \mathbb{R}^D$, and $\sigma^2 > 0$. Tipping and Bishop (1999) showed that for $\sigma^2 \to 0$ the model recovers the standard PCA (but the components of $\mathbf{z}$ might be permuted).

We assume that the data $\mathbf{x}_1, \ldots, \mathbf{x}_N$ is identically and independently distributed according to the PPCA model. In a maximum likelihood setting, one determines the parameters $\mathbf{W}$, $\boldsymbol{\mu}$, and σ^2 that maximize the likelihood

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{W}, \boldsymbol{\mu}, \sigma^2) = \prod_{n=1}^{N} p(\mathbf{x}_n; \mathbf{W}, \boldsymbol{\mu}, \sigma^2),$$

or equivalently the log-likelihood

$$\log p(\mathbf{x}_1, \dots, \mathbf{x}_N; \mathbf{W}, \boldsymbol{\mu}, \sigma^2) = \sum_{n=1}^{N} \log p(\mathbf{x}_n; \mathbf{W}, \boldsymbol{\mu}, \sigma^2).$$

Question 3.3: *Show that for the model of the probabilistic PCA*

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{C}),$$

where $\mathbf{C} = \mathbf{W}\mathbf{W}^\intercal + \sigma^2 \mathbf{I}_D$.

Tip 3.1 Use Corollary 2 in Lecture 2. ○

Answer:

The PPCA is defined as a model for how to decode a latent vector $\mathbf{z}$ to the observed data $\mathbf{x}$. Of course, as in the regular PCA, we would like to encode data $\mathbf{x}$ in the lower dimensional latent space as well.

Question 3.4: *Show that the distribution of the latent variable* $\mathbf{z}$ *conditioned on* $\mathbf{x}$ *is Gaussian as well and given by*

$$p(\mathbf{z} \,|\, \mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \mathbf{M}^{-1}\mathbf{W}^{\intercal}(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}^{-1}\right),$$

where $\mathbf{M} = \mathbf{W}^{\intercal}\mathbf{W} + \sigma^2\mathbf{I}_M$.[2]

Tip 3.2 Use Corollary 1 in Lecture 2. ○

Answer:

[2]Note that eq. (12.42) in Bishop's book is incorrect. In the original paper by Tipping and Bishop (1999) the correct distribution is stated.

3.3 Variational autoencoder

Let us consider a more flexible nonlinear model that is given by

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) &= \mathcal{N}\left(\mathbf{x}; \mu_{\boldsymbol{\theta}}(\mathbf{z}), \sigma_{\boldsymbol{\theta}}^2 \mathbf{I}_D\right), \\ p_{\boldsymbol{\theta}}(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}_M), \end{aligned}$$

where parameters $\boldsymbol{\theta}$ include variance parameter $\sigma_{\boldsymbol{\theta}}^2 > 0$ and $\mu_{\boldsymbol{\theta}} \colon \mathbb{R}^M \to \mathbb{R}^D$ is a nonlinear model of the mean of the normal distribution. In this lab we will model $\mu_{\boldsymbol{\theta}}$ by a neural network and include its weights and biases in $\boldsymbol{\theta}$, but other models could be used equally well.

In the lab session we will discuss different training approaches for this model. One approach involves defining an encoding distribution $q_{\boldsymbol{\phi}}(\mathbf{z}; \mathbf{x})$ that may depend on $\mathbf{x}$ and some parameters $\boldsymbol{\phi}$, and estimating and maximizing the so-called evidence lower bound (ELBO)

$$\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z};\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}\big(q_{\boldsymbol{\phi}}(\mathbf{z}; \mathbf{x}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{z})\big).$$

A common choice is to define q as a multivariate normal distribution with diagonal covariance matrix, i.e., to set

$$q_{\boldsymbol{\phi}}(\mathbf{z}; \mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_{\boldsymbol{\phi}}(\mathbf{x}), \operatorname{diag}(\sigma_{\boldsymbol{\phi}}^2(\mathbf{x}))), \tag{3}$$

where $\mu_{\boldsymbol{\phi}} \colon \mathbb{R}^D \to \mathbb{R}^M$ defines the mean of the normal distribution and $\sigma_{\boldsymbol{\phi}}^2 \colon \mathbb{R}^D \to \mathbb{R}_{\geq 0}^M$ the diagonal entries of the covariance matrix. We will model these functions with a neural network, but, of course, other choices are possible as well. For this particular choice of the encoding distribution q a closed-form expression of the KL divergence term exists.

As shown in Exercise 6.1, for the distribution $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}_M)$ of the nonlinear model and the choice of distribution $q_{\boldsymbol{\phi}}(\mathbf{z}; \mathbf{x})$ in eq. (3) we have

$$\begin{aligned} &\mathrm{KL}\big(q_{\boldsymbol{\phi}}(\mathbf{z}; \mathbf{x}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{z})\big) \\ &\qquad = \frac{1}{2}\bigg(\sum_{i=1}^M \left(\sigma_{\boldsymbol{\phi}}^2(\mathbf{x})\right)_i + \left\|\mu_{\boldsymbol{\phi}}(\mathbf{x})\right\|_2^2 - M - \sum_{i=1}^M \log\left(\sigma_{\boldsymbol{\phi}}^2(\mathbf{x})\right)_i\bigg). \end{aligned}$$

4 Laboratory session

This section contains instructions for the laboratory session. The main lab exercise is to build and train models that generate images that look like hand-written digits. The generation will be made possible by unsupervised learning of the most important features of such images from a data set presented in Section 4.1.

4.1 MNIST data set

We consider the so-called MNIST data set, which is one of the most well-studied data sets within machine learning and image processing.

The data set consists of 60 000 training data points and 10 000 test data points. Each data point consists of a grayscale image with 28×28 pixels of a hand-written digit. The digit has been size-normalized and centered within a fixed-sized image. Each image is also labeled with the digit (0, 1, . . . , 8, or 9) it is depicting. In Figure 1 a batch of 25 data points from this data set is displayed.

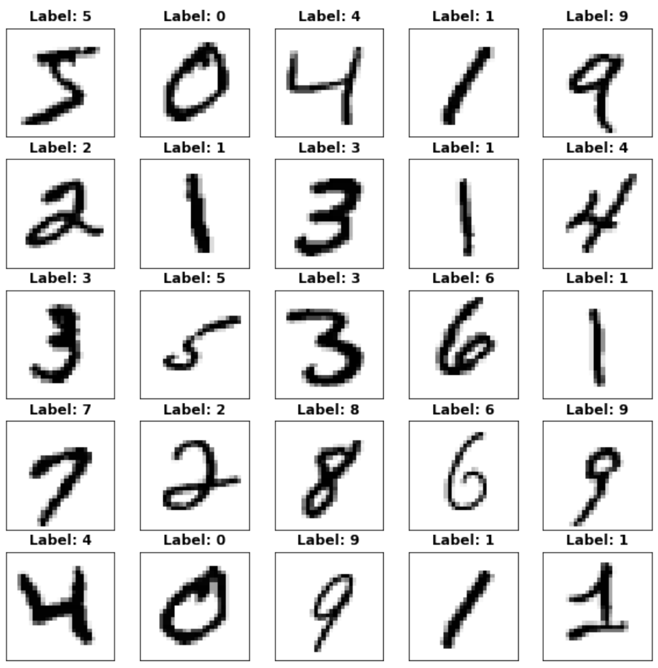

Figure 1: Samples from the MNIST data set.

We consider each image as a vector $\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix}^\mathsf{T}$, where each x_j corresponds to one of the $p = 28 \times 28 = 784$ pixels in the image. The value of each x_j represents the color of that pixel. The color value is within the interval $[0, 1]$, where $x_j = 0$ corresponds to a black pixel and $x_j = 1$ to a white pixel. Anything between 0 and 1 is a gray pixel with corresponding intensity.

Based on a set of training data $\{\mathbf{x}_i\}_{i=1}^N$ with images, the problem is to find a good model that allows us to sample from the distribution

$$p(\mathbf{x})$$

of the images. Unfortunately, it is unclear what the best way is to model this distribution of 784-dimensional vectors, since probably most of these vectors do not even represent images of hand-written digits.

We approach the problem by trying to extract all important features of the images and compress them to a low dimensional space, in such a way that we can reconstruct the images reasonably well. Instead of generating new images by sampling 784-dimensional vectors, we then try to sample a new set of low dimensional image features and to construct a proper image from it.

4.2 Preparation

Download the file `lab_code.zip` from studium and extract it. It contains a set of Jupyter notebooks which we will work on in the laboratory session.

4.3 Principal component analysis

One of the most common tools for dimensionality reduction is PCA. We start by performing a PCA on the MNIST data.

Task 4.1 Open the Jupyter notebook `PCA.ipynb` and work through it. ◦

Answer the following questions as part of Task 3 in the notebook.

Question 4.1: *Which digits can be reconstructed and decoded quite well, and which ones seem to be more challenging?*

Answer:

Question 4.2: *What average squared reconstruction error do you get with PCA?*

Answer:

4.4 Probabilistic principal component analysis

Since we want to sample from the distribution of the low dimensional image features, we move to a probabilistic setting and include a probabilistic description of these encodings in our model.

Task 4.2 Open the the Jupyter notebook `Probabilistic PCA.ipynb` and work through it. ○

Answer the following question as part of Task 2 in the notebook.

Question 4.3: *How were the parameters of the model initialized? For how many iterations was the model trained?*

Answer:

Answer the following questions as part of Task 4 in the notebook.

Question 4.4: *Compare the encoding of the MNIST data set with PCA and PPCA. Can you explain the different ranges of the encodings?*

Answer:

Question 4.5: *Which digits can be reconstructed and decoded quite well, and which ones seem to be more challenging? Compare your answer with Question 4.1.*

Answer:

Question 4.6: *What average squared reconstruction error do you get with PPCA? Compare your answer with Question 4.2.*

Answer:

4.5 Variational autoencoder

Task 4.3 Open the Jupyter notebook `VAE.ipynb` and work through it. ○

Answer the following question as part of Task 1 in the notebook.

Question 4.7: *Have another look at the mean encodings of the digits 0, 1, . . . , 9 with PCA and PPCA. For which digits do you observe encodings close to each* *other? Compare your answer with Questions 4.1 and 4.5.*

Answer:

Answer the following questions as part of Task 8 in the notebook.

Question 4.8: *Compare the encoding of the MNIST data set with the VAE and PPCA. What do you notice?*

Answer:

Question 4.9: *Which digits can be reconstructed and decoded quite well, and which ones seem to be more challenging? Compare your answer with Question 4.5.*

Answer:

Question 4.10: *What average squared reconstruction error do you get with the VAE? Compare your answer with Question 4.6.*

Answer:

References

Kingma, D. P., & Welling, M. (2019). *An introduction to variational autoencoders*. arXiv: 1906.02691 [cs.LG].

Tipping, M. E., & Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *61*(3), 611–622.